Practise Session 2 : Working with Docker Container Engine

Make sure that you have already gone through Lab-01

Docker is an open-source platform for developing, shipping, and running applications. Docker enables you to separate applications from infrastructure and deliver your applications quickly from sandbox to production environments. DevOps is primarily used to overcome 'Dev' and 'Ops' problems, and Docker seems to solve most of them by providing seamless control over all inevitable changes in development, production, and staging environments. So, the aim of an experiment is to get acquainted with docker fundamentals to advanced features such as to build and push the container images.

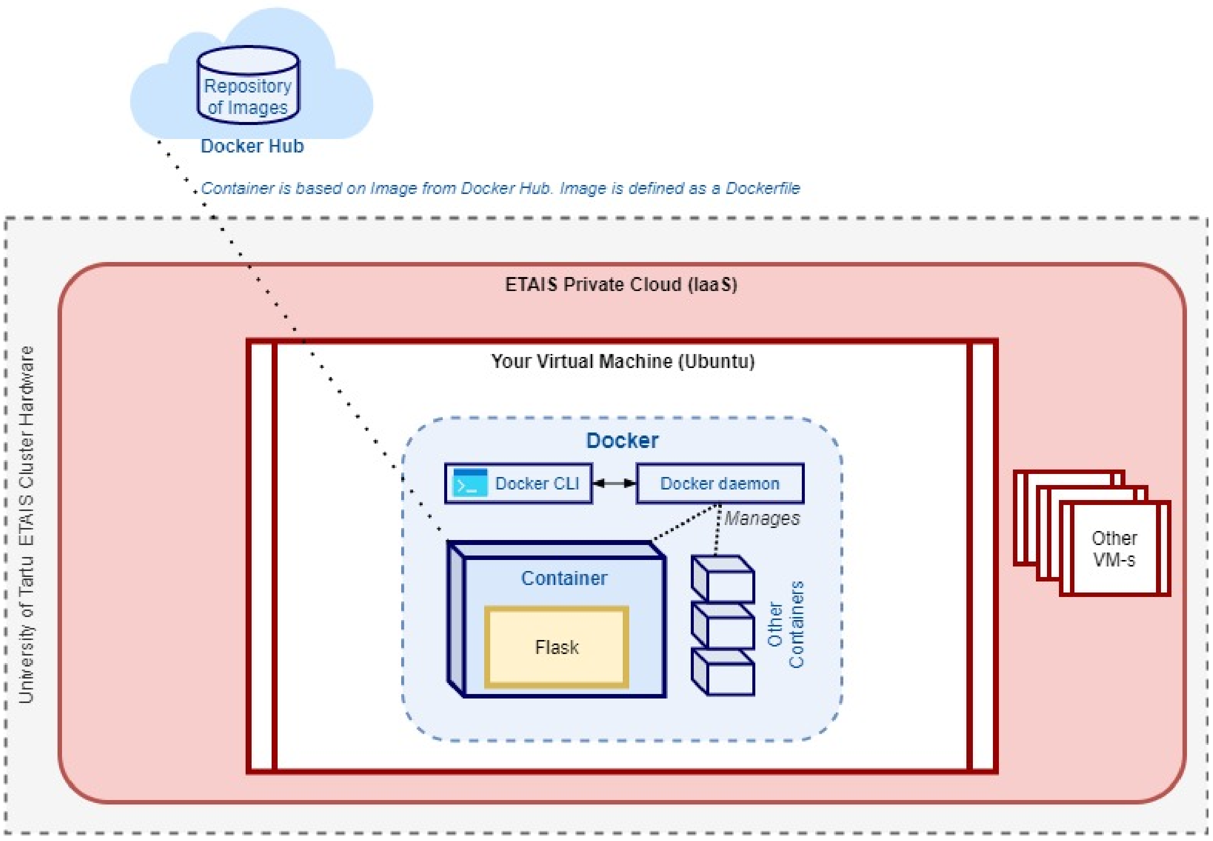
Introduction to Docker

Docker containers allow the packaging of your application (and everything that you need to run it) in a "container image". Inside a container, you can include a base operating system, libraries, files and folders, environment variables, volume mount-points, and your application binaries.

- Docker image
  - Lightweight, standalone, executable package that includes everything needed to run a piece of software
  - Includes code, a runtime, library, environment variables, and config files
- Docker container
  - Runtime instance of an image - what the image becomes in memory when actually executed.
  - Completely isolated from the host environment.

Build, Ship, and Run Any App, Anywhere

The below Figure shows the architecture of docker installation in the ETAIS cloud and its interaction with the Docker hub.


Repository of Images
Docker Hub
Container is based on Image from Docker Hub. Image is defined as a Dockerfile
University of Tartu ETAIS Cluster Hardware
ETAIS Private Cloud (IaaS)
Your Virtual Machine (Ubuntu)
Docker
Docker CLI
Docker daemon
Manages
Container
Flask
Other Containers
Other VM-s

- Docker is available in two editions: Community Edition (CE) and Enterprise Edition (EE)
- Supported Platform: macOS, Microsoft Windows 10, CentOS, Debian, Fedora, RHEL, Ubuntu, and more on https://store.docker.com/.

References

Referred documents and websites contain supportive information for the practice.

Manuals

1. Docker fundamentals
2. Docker CLI
3. Building docker image
4. Docker swarm
5. Docker architecture

In case of issues check:

1. Possible solutions to common issues section at the end of the guide.
2. Ask in the `#Lab02-Docker` topic in zulip.

Prerequisites

- Make sure that you have already gone through Lab-01
- Create a login account in docker hub using this link and sign up.

Exercise 1. Installation of docker in an ETAIS Virtual Machine

In this task, you are going to install docker on your VM that you have created in the last practice session. Try to run the basic commands to make yourself comfortable with docker.

- Make sure that you are able to access (preferably over ssh) your VM created in Lab-01.Connect to the virtual machine remotely using gitbash or any other SSH client.
    - `NB! Extra modifications required to change docker network settings:`
- Create a directory in the virtual machine in the path: `sudo mkdir /etc/docker`
- Create a daemon file in the docker directory: `sudo vi /etc/docker/daemon.json`
- Copy the following script:

```
{
    "default-address-pools": [{"base":"172.80.0.0/16","size":24}]
}
```

- Now run this command. It will add the official Docker repository, download the latest version of Docker, and install it:
```
curl -fsSL https://get.docker.com/ | sh
```
- After installation has completed, start the Docker daemon:
```
sudo systemctl start docker
```
- Verify that Docker service is running:
```
sudo systemctl status docker
```

`NB! To run docker commands with non-root privileges`

- Create a docker group (If it's already created then ignore): `sudo groupadd docker`
- Add a user to docker group: `sudo usermod -aG docker $USER`
- Activate the changes: `newgrp docker`
- Check the installation by displaying docker version: `docker --version`

Exercise 2. Practising docker commands

This task mainly helps you to learn basic commands used by docker CLI such as run, pull, listing images, attaching data volume, working with exec (like ssh a container), checking IP address, and port forwarding. Docker commands

- **Pull an image from Docker Hub and run an Ubuntu container** in detached mode and assign your name as container-name, install HTTP server (use `docker exec` command) and use port forwarding to access container-http traffic via host port 80.
    - Create a login account at Docker Hub sign-up page (Prerequisite as mentioned earlier)
    - Login into your docker account from docker host terminal: `docker login`
        - Input the following:
            - Username: your docker hub id
            - Password: Docker hub password

- NB! The login part is not mandatory but needed/recommended since recently docker hub limited the number of docker pull from a particular IP. As you are using the university network through VPN, it serves as a single IP for the Docker hub. (Error response from daemon: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit)
- Pull an image from docker hub: `docker pull ubuntu`
- Check the downloaded images in local repository: `docker images`
- Run a simple ubuntu container: `docker run -dit -p 80:80 --name <new_container_name> ubuntu`
    - <new_container_name> = please type any name for that container
- Get the bash shell of container: `docker exec -it <container_name> sh`
- Exit from the container: `exit`
- Connect to container and update the apt repo: `docker exec -dit <container_name> apt-get update`
- Install http server: `docker exec -it <container_name> apt-get install apache2`
- Check the status of http server: `docker exec -it <container_name> service apache2 status`
- If not running, start the http server: `docker exec -it <container_name> service apache2 start`
- Check the webserver running container host machine: `curl localhost:80`
- Check the ip address of the container: `docker inspect <container_name> | grep -i "IPAddress"`

- **Host directory as a data volume:** Here you are mounting a host directory in a container and this is useful for testing the applications. For example, if you store source code in the host directory and mount it in the container, the code changed in the host directory file can affect the application running in the container.
    - Accessing a host file system on the container with read-only and read/write modes:
        - Create directory in home directory (*/home/centos*) with name test: `mkdir test && cd test`, Create a file: `touch abc.txt`
        - Run a container with a `-v` parameter to mount the host directory to the container
            - Read only: `docker run -dit -v /home/centos/test/:/home/:ro --name vol1 ubuntu sh`

- Access the file in a container in the path /home and try to create a new file(you should see access denied) from container `docker exec -it vol1 sh`, `cd /home`, `ls`, `exit`.
- Read/write: `docker run -dit -v /home/centos/test/:/home/:rw --name vol2 ubuntu` , `docker exec -it vol2 sh` , `cd /home` , `ls` . Try to create some text files,exit. You can see the created files in host machine `cd /home/centos/test/` , `ls`
- Stop and delete the container : `docker stop vol1` , `docker rm vol1`

- **Data volume containers:** A popular practice with Docker data sharing is to create a dedicated container that holds all of your persistent shareable data resources, mounting the data inside of it into other containers once created and set up.
  - Create a data volume container and share data between containers.
    - Create a data volume container `docker run -dit -v /data --name data-volume ubuntu` , `docker exec -it data-volume sh`
    - Go to volume and create some files : `cd /data && touch file1.txt && touch file2.txt` . Exit the container with the `exit` command.
    - Run another container and mount the volume of earlier container : `docker run -dit --volumes-from data-volume --name data-shared ubuntu` , `docker exec -it data-shared sh`
    - Go to the data directory in the created container and list the files: `cd /data && ls`

Screenshot - 1

NB! Complete the below mentioned scenario and take a screenshot of the outputs.

- **Scenario** : Containerize a program to display your name, hostname, and IP address of the container. Use any of your favourite programming languages. The expected output is to display the information in the terminal.

Exercise 3. Building a Dockerfile

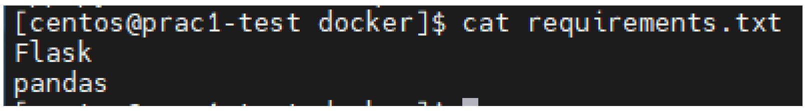
The task is to create your own docker image and try to run the same. More information about Dockerfile (https://docs.docker.com/engine/reference/builder/). A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

Dockerfile commands:

FROM	Set the base image
LABEL	Add a metadata to image

RUN	Execute the command and execute the commands
CM	Allowed only once
EXPOSE	Container listen on the specific ports at runtime
ENV	Set environment variables
COPY	Copy the files or directories into container's filesystem
WORKDIR	Set working directory

- **The scenario is to deploy a web application (flask) to read a IoT sensor data from csv and display it on the web page. This is a continuation task of what you did in** Lab-01, *Exercise 2.4*.
    - `cd flask_app`
    - Inside "`flask_app`" directory you should have following files and folder:
        - `app.py`
        - `CO2.csv`
        - `templates`
            - `home.html`
    - We will not include `CO2.csv` data file in the image building process. It's better to move all the related files and folders to another directory (say `docker_app`).
        - `mkdir docker_app`
        - `mv app.py docker_app`
        - `mv templates docker_app`
        - `cd docker_app`
    - Now you are inside the `$HOME/flask_app/docker_app` directory. Create a requirement file to install libraries required to run the Flask application `requirements.txt` and add the text **Flask** and **pandas** in the file.



```
[centos@prac1-test docker]$ cat requirements.txt
Flask
pandas
```

    - Create Docker file to build a container image using the Dockerfile commands `sudo vi Dockerfile`

```
FROM python:3.8-slim-buster

WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

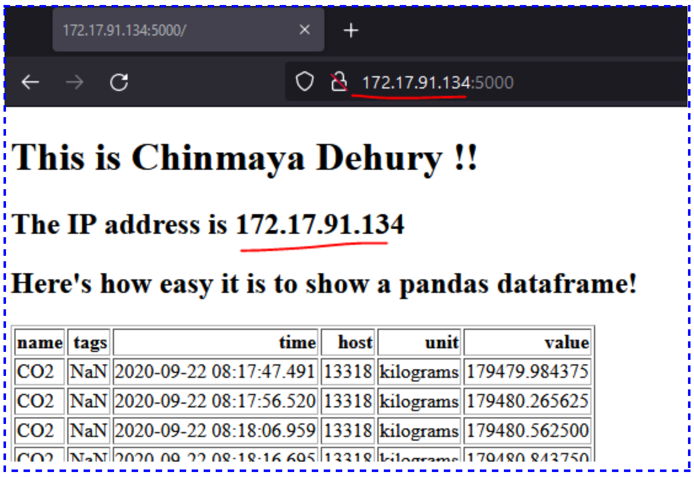
- Build the docker file: `docker build -t myflask .`
  Don't forget the dot at the end of the build command
- Check the created image in the repository list : `docker images`
- Run the container with the created image. Here we need to mount the data file (i.e. `CO2.csv`) which is outside the current `docker` directory:
    - `docker run -d -p 5000:5000 -v "$(pwd)/../CO2.csv":"/app/CO2.csv" myflask`
- Check the running container at http://Your_VM_IP:5000/

NB! You should see your last modifications (your name and IP address) along with sensor data.

Screenshot - 2

Take a screenshot of the webpage.

Screenshot example below:


172.17.91.134:5000/

172.17.91.134:5000

This is Chinmaya Dehury !!

The IP address is 172.17.91.134

Here's how easy it is to show a pandas dataframe!

name	tags	time	host	unit	value
CO2	NaN	2020-09-22 08:17:47.491	13318	kilograms	179479.984375
CO2	NaN	2020-09-22 08:17:56.520	13318	kilograms	179480.265625
CO2	NaN	2020-09-22 08:18:06.959	13318	kilograms	179480.562500
CO2	NaN	2020-09-22 08:18:16.695	13318	kilograms	179480.843750

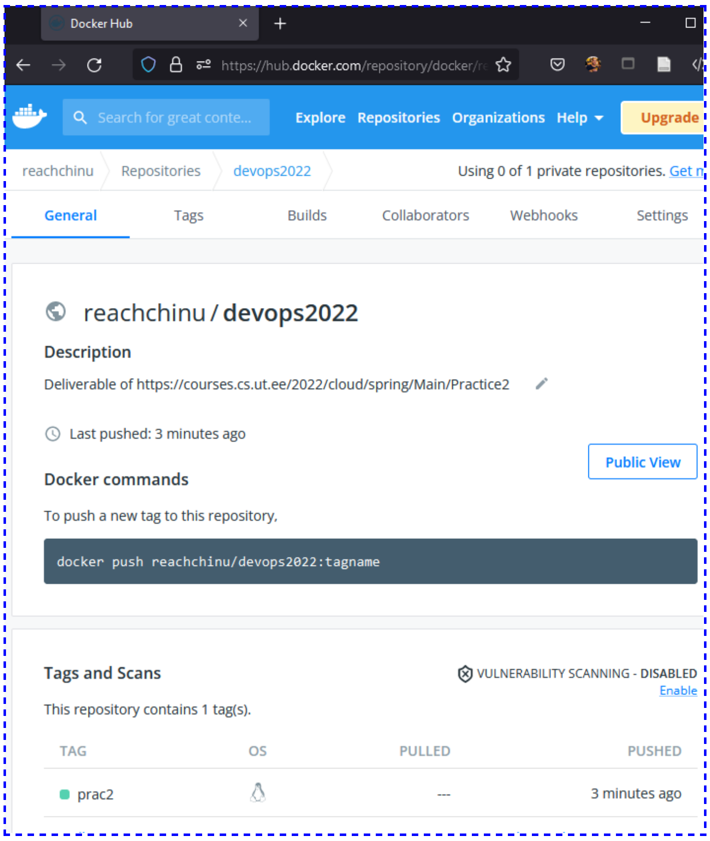
Exercise 4. Shipping a docker image to Docker hub

Docker hub is a hosted repository service provided by Docker for finding and sharing container images with your team.

- Create a login account in docker hub using this link and sign up.
- **Push the docker image to docker hub**
  - Initially login into your docker account from docker host terminal: `docker login`
    - Provide input to the following:
      - Username: your docker hub id
      - Password: Docker hub password
  - Tag your image before pushing into docker hub :
    - `docker tag myflask your_docker_username/devops2022:prac2`
  - Finally push the image to docker hub:
    - `docker push your_docker_username/devops2022:prac2`

Screenshot - 3

NB! Take the screenshot of the Docker hub account showing your image.


Docker Hub
https://hub.docker.com/repository/docker/r
Search for great conte...
Explore Repositories Organizations Help
Upgrade
reachchinu Repositories devops2022
Using 0 of 1 private repositories. Get n
General Tags Builds Collaborators Webhooks Settings
Public View
VULNERABILITY SCANNING - DISABLED
Enable

reachchinu / devops2022

Description

Deliverable of https://courses.cs.ut.ee/2022/cloud/spring/Main/Practice2

Last pushed: 3 minutes ago

Docker commands

To push a new tag to this repository,

```
docker push reachchinu/devops2022:tagname
```

Tags and Scans

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
prac2		---	3 minutes ago

Bonus Task:

Visualise sensor data in Grafana: containerizing an application.

In this use case, you are going to perform following tasks:

- Creating a docker volume.
- Creating a container to download and store the data in the docker volumes.
- Creating a container with Grafana and read the sensor data from the docker volumes and display it using Grafana.

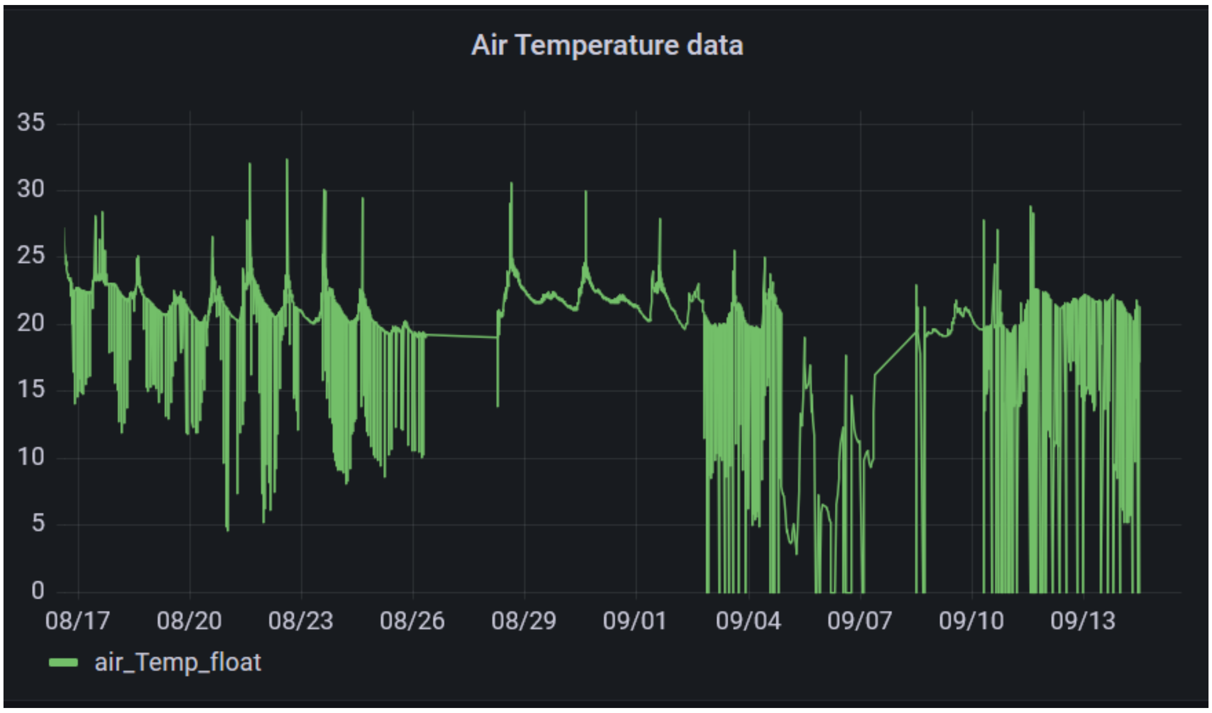
NB! This exercise should be completed by yourself using the knowledge from the previous exercise and its bonus task (not compulsory exercise)

- Creating a docker volume. You may use `docker volume create <vol_name>` command.
- Create a Dockerfile that should download the sensor data (you can use `git clone https://github.com/shivupoojar/devops-prac2.git`) and run the container that stores this data in the docker volumes.
  NB! The above task can be done in different ways, the following is an example.
  - Use base image as `ubuntu`
  - Make working directory as `/app` using WORKDIR
  - Define ARG as argument variable git_url(You can use git project url https://github.com/shivupoojar/devops-prac2.git as argument while building an image)
  - Update the apt-get

  - Install git using apt
  - Clone the project (use ARG variable containing project url) containing iot sensor data to build image you can use
    - `docker build -t image_name --build-arg git_url="https://github.com/shivupoojar/devops-prac2.git" .`
  - Run the container as
    - `docker run --name name -d --mount source=volume_name destination=/app image_name`
- Now let us create a Dockerfile for grafana image as shown below and build it. Here you should use the Grafana docker image and install grafana plugin for reading CSV (for more information please refer to this).

```
FROM grafana/grafana
RUN grafana-cli plugins install marcusolsson-csv-datasource
```

- Built the image using docker build command
- Run the container with the following options:
  - use port mapping 3000:3000 (Attach `allow-all` security group to your VM to access the web app through port `3000`)
  - link the docker volume to access sensor data using **mount** option (can refer here)
- In Grafana, we are using local file path as a data source (.csv file) for that you modify the grafana.ini configuration file as **Local mode**
  - To allow the **Local mode** you need to update grafana.ini file located at **/etc/grafana/grafana.ini** and to do this, already updated grafana.ini can be found in mounted directory and can be copied to **/etc/grafana/grafana.ini** using **cp** command (Use docker exec and command to use is `cp path/grafana.ini /etc/grafana/grafana.ini`). You may need to use `--user root` flag while executing `docker exec` command.
  - After doing this, restart the container and again open the grafana dashboard and continue.
- After running the container, you should access the grafana using http://VM_IP:3000 and login using **username** as **admin** and **Password** as **admin** and skip for password update.
- Now let us add a data source - Click on Configuration tab --> Data Source --> CSV (search for data source as CSV).
  - Enable **Local**
  - Path --> **/usr/share/grafana** (this should same as you specify mount name while attaching docker volume to the container)
  - Click on Save and Test
- Creating a dashboard
  - Click on **create (+)** --> dashboard than **Add an empty panel**
  - You can refer to this manual to create a dashboard
    - Path-->airdata.csv
    - Fields--> time (time format) and air_Temp_float(Number format)
    - Change the time range 30 days and you should see the data in grafana as shown below (You can design nice dashboard even though :-))



Screenshot - 4

NB! Take the screenshot of the grafana dashboard with a clearly visible VM IP address .

Deliverables

1- Gather all the screenshots

- Screenshot 1
- Screenshot 2
- Screenshot 3
- Screenshot 4 (Bonus Task- Not compulsory exercise)

2- zip the flask_app directory and all the screenshots

3- Upload the zip file to the course wiki page.

4- ***[Note]*** Do not delete this zip file. You need this in **Lab03.**

5- **Stop** the Virtual Machine and you can start using the same in **Lab03.**

Don't delete your VM

References:

- List of recommended Books: https://docs.docker.com/get-started/resources/#books
- Online learning: https://docs.docker.com/get-started/resources/#books